|| Jai Sri Gurudev ||

Sri Adichunchanagiri Education Trust (R)

S J C Institute of Technology

(VTU affiliated, AICTE approved, and NBA and NAAC Accredited)

PB. No. 20, BB Road, Chickballapur-562 101

MERN (BDSL456C)

FOR 4TH SEMESTER

Prepared by

Mr. Yashwantha N

Assistant Professor

Department of Artificial Intelligence and Data Science

SJCIT

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SJC INSTITUTE OF TECHNOLOGY

B.B. ROAD CHICKBALLAPUR – 562101

2023-2024

MERN

Sl. No.	Experiments
01	Create a simple resume using HTML that includes sections for personal information, education, work experience, and skills. Ensure that each section is clearly labelled and styled appropriately.
02	a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection. b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]
03	Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the query] a. Find any record where Name is Somu b. Find any record where total payment amount (Payment.Total) is 600. c. Find any record where price (Transaction.price) is between 300 to 500. d. Calculate the total transaction amount by adding up Payment.Total in all records.
04	Implement all CRUD operations on a File System using Node JS
05	a. Read the data of a student containing usn, name, sem, year_of_admission from node js and store it in the mongodb b. For a partial name given in node js, search all the names from mongodb student documents created in Question(a)
06	Develop the application that sends fruit name and price data from client side to Node.js server using Ajax
07	Develop a React code to build a simple search filter functionality to display a filtered list based on the search query entered by the user.
08	Develop two routes: find_prime_100 and find_cube_100 which prints prime numbers less than 100 and cubes less than 100 using Express JS routing mechanism.
09	Develop an authentication mechanism with email_id and password using HTML and Express JS (POST method)
10	

MERN

4th Sem

1. **Create a simple resume using HTML that includes sections for personal information, education, work experience, and skills. Ensure that each section is clearly labelled and styled appropriately.**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Resume of Jane Doe</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
      background-color: #f9f9f9;
    }
    .container {
      max-width: 800px;
      margin: auto;
      background: white;
      padding: 20px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    h1, h2 {
      text-align: center;
    }
    .section {
      margin-bottom: 20px;
    }
    .section h2 {
      border-bottom: 2px solid #ddd;
      padding-bottom: 5px;
    }
    .section ul {
      list-style-type: none;
      padding: 0;
    }
    .section ul li {
      margin-bottom: 10px;
    }
  </style>
```

```
</head>
<body>
  <div class="container">
    <h1>Yashwanth N</h1>
    <p>Email: yashwanth.nagaraj@gmail.com | Phone: +91-7795373834 | LinkedIn: https://www.linkedin.com/in/yashwantha-nagaraj-6400a0141/</p>

    <div class="section">
      <h2>Objective</h2>
      <p>Motivated and enthusiastic computer science graduate seeking an entry-level software development position to leverage my skills in coding, problem-solving, and teamwork to contribute to the growth and success of a dynamic organization.</p>
    </div>

    <div class="section">
      <h2>Education</h2>
      <ul>
        <li><strong>Bachelor of Engineering in Computer Science and Engineering</strong><br>SJB Institute of Technology, Bengaluru</li>
        <li><strong>Master of Technology in Computer Science and Engineering</strong><br>RNS Institute of Technology, Bengaluru</li>
      </ul>
    </div>

    <div class="section">
      <h2>Projects</h2>
      <ul>
        <li><strong>Portfolio Website</strong><br>Developed a personal portfolio website using HTML, CSS, and JavaScript to showcase projects and skills.</li>
        <li><strong>Library Management System</strong><br>Created a library management system using Python and MySQL to manage book inventories and user transactions.</li>
        <li><strong>Weather App</strong><br>Built a weather application using React and OpenWeatherMap API to display current weather conditions based on user location.</li>
      </ul>
    </div>

    <div class="section">
      <h2>Skills</h2>
      <ul>
        <li>Proficient in HTML, CSS, JavaScript</li>
        <li>Experience with Python and Java</li>
        <li>Knowledge of React.js and Node.js</li>
        <li>Strong problem-solving and analytical skills</li>
        <li>Excellent communication and teamwork abilities</li>
      </ul>
    </div>

    <div class="section">
```

```
        <h2>Certifications</h2>
        <ul>
          <li>Certified Java Developer - Oracle</li>
          <li>Front-End Web Developer Nanodegree - Udacity</li>
        </ul>
        <ul>
          <li>Certified Full Stack Developer - Oracle</li>
          <li>Web Developer Nanodegree - Udemy</li>
        </ul>
      </div>

      <div class="section">
        <h2>Extracurricular Activities</h2>
        <ul>
          <li>Member of the Computer Science Club at VTU</li>
          <li>Volunteer at local coding bootcamps for underprivileged youth</li>
          <li>Participant in hackathons and coding competitions</li>
        </ul>
      </div>
    </div>
</body>
</html>
```

Output:

Yashwanth N

Email: yashwanth.nagaraj@gmail.com | Phone: +91-7795373834 | LinkedIn: https://www.linkedin.com/in/yashwantha-nagaraj-6400a0141/

Objective

Motivated and enthusiastic computer science graduate seeking an entry-level software development position to leverage my skills in coding, problem-solving, and teamwork to contribute to the growth and success of a dynamic organization.

Education

Bachelor of Engineering in Computer Science and Engineering
SJB Institute of Technology, Bengaluru

Master of Technology in Computer Science and Engineering
RNS Institute of Technology, Bengaluru

Projects

Portfolio Website
Developed a personal portfolio website using HTML, CSS, and JavaScript to showcase projects and skills.

Library Management System
Created a library management system using Python and MySQL to manage book inventories and user transactions.

Weather App
Built a weather application using React and OpenWeatherMap API to display current weather conditions based on user location.

Skills

Proficient in HTML, CSS, JavaScript

Experience with Python and Java

Knowledge of React.js and Node.js

Strong problem-solving and analytical skills

Excellent communication and teamwork abilities

Certifications

Certified Java Developer - Oracle

Front-End Web Developer Nanodegree - Udacity

Certified Full Stack Developer - Oracle

Web Developer Nanodegree - Udemy

Extracurricular Activities

Member of the Computer Science Club at VTU

Volunteer at local coding bootcamps for underprivileged youth

Participant in hackathons and coding competitions

2 a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.

b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]

2a.

```
test> use MoviesDB

switched to db MoviesDB

MoviesDB> db.createCollection("Movies")

{ ok: 1 }

MoviesDB> db.Movies.insertMany([

  { title: "Inception", director: "Christopher Nolan", genre: "Science Fiction", year: 2010, ratings: { imdb: 8.8, rottenTomatoes: 87 } },

  { title: "The Matrix", director: "Wachowskis", genre: "Science Fiction", year: 1999, ratings: { imdb: 8.7, rottenTomatoes: 87 } },

  { title: "The Godfather", director: "Francis Ford Coppola", genre: "Crime", year: 1972, ratings: { imdb: 9.2, rottenTomatoes: 97 } }

]);

MoviesDB> db.Movies.find({}, { title: 1, director: 1 })

MoviesDB> db.Movies.find({}, { title: 1, director: 1, _id: 0 })

MoviesDB> db.Movies.find({}, { ratings: 0 })

MoviesDB> db.Movies.find({ director: "Christopher Nolan" }, { title: 1, year: 1, _id: 0 })
```

2b. Use of limit and find in MongoDB query

```
test> use MoviesDB

MoviesDB> db.createCollection("Movies")

MoviesDB>db.Movies.insertMany([
```

```
  { title: "Inception", director: "Christopher Nolan", genre: "Science Fiction", year: 2010, ratings: { imdb: 8.8, rottenTomatoes: 87 } },
  { title: "The Matrix", director: "Wachowskis", genre: "Science Fiction", year: 1999, ratings: { imdb: 8.7, rottenTomatoes: 87 } },
  { title: "The Godfather", director: "Francis Ford Coppola", genre: "Crime", year: 1972, ratings: { imdb: 9.2, rottenTomatoes: 97 } },
  { title: "Pulp Fiction", director: "Quentin Tarantino", genre: "Crime", year: 1994, ratings: { imdb: 8.9, rottenTomatoes: 92 } },
  { title: "The Shawshank Redemption", director: "Frank Darabont", genre: "Drama", year: 1994, ratings: { imdb: 9.3, rottenTomatoes: 91 } },
  { title: "The Dark Knight", director: "Christopher Nolan", genre: "Action", year: 2008, ratings: { imdb: 9.0, rottenTomatoes: 94 } },
  { title: "Fight Club", director: "David Fincher", genre: "Drama", year: 1999, ratings: { imdb: 8.8, rottenTomatoes: 79 } }
]);
```

Query with Projection and Limit

```
MoviesDB> db.Movies.find({}, { title: 1, director: 1, year: 1, _id: 0 }).limit(5)
```

MERN

3. Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the query]

a. Find any record where Name is Somu

b. Find any record where total payment amount (Payment.Total) is 600.

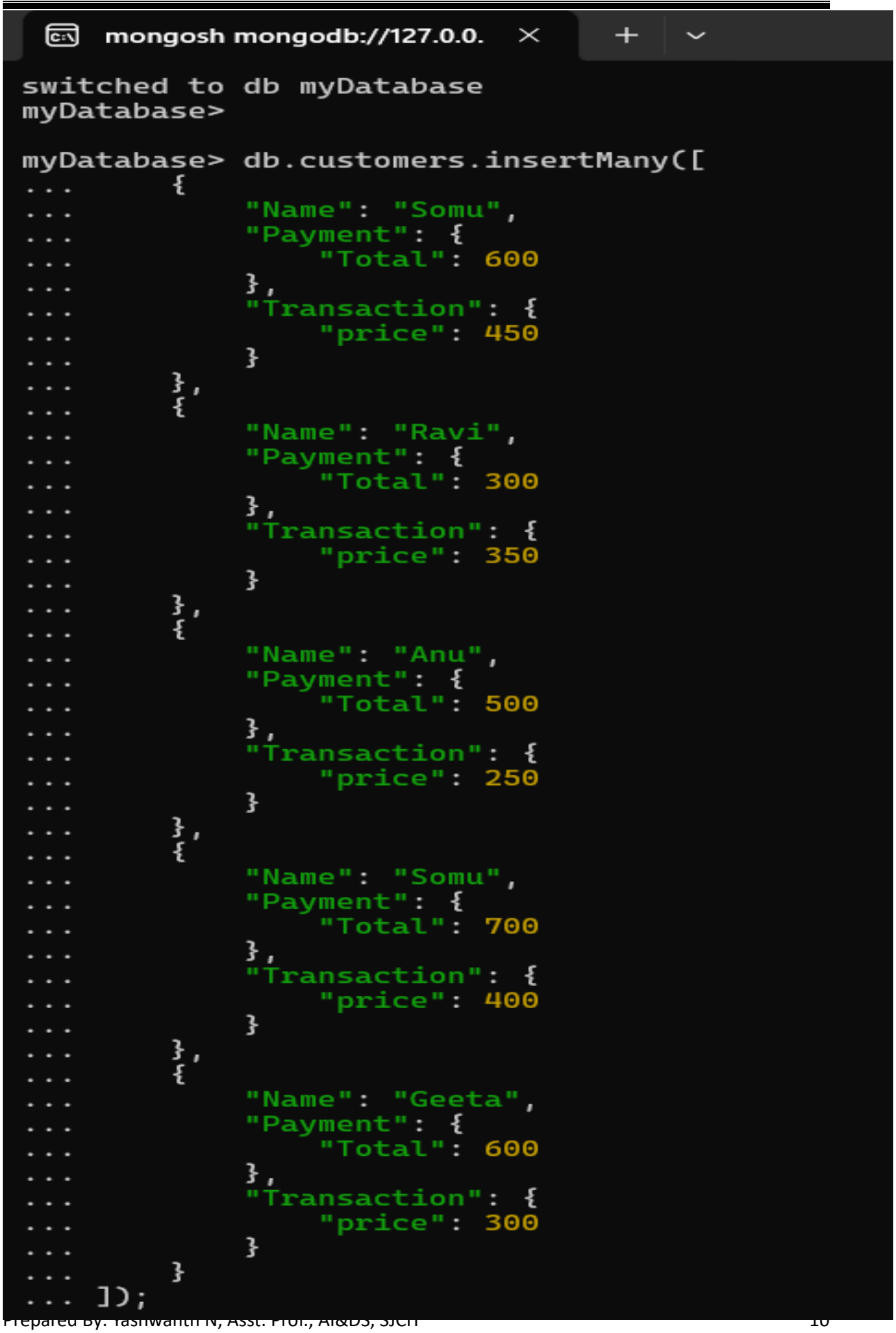
c. Find any record where price (Transaction.price) is between 300 to 500.

d. Calculate the total transaction amount by adding up Payment.Total in all records.

Step 1: Create the Collection and Insert Documents.

Use myDatabase;

MERN


mongosh mongodb://127.0.0.
switched to db myDatabase
myDatabase>
myDatabase> db.customers.insertMany([
{
"Name": "Somu",
"Payment": {
"Total": 600
},
"Transaction": {
"price": 450
}
},
{
"Name": "Ravi",
"Payment": {
"Total": 300
},
"Transaction": {
"price": 350
}
},
{
"Name": "Anu",
"Payment": {
"Total": 500
},
"Transaction": {
"price": 250
}
},
{
"Name": "Somu",
"Payment": {
"Total": 700
},
"Transaction": {
"price": 400
}
},
{
"Name": "Geeta",
"Payment": {
"Total": 600
},
"Transaction": {
"price": 300
}
}
]);

Step 2: Queries

a. Find any record where Name is Somu

```
myDatabase> db.customers.find({ "Name": "Somu" });
```

b. Find any record where total payment amount (Payment.Total) is 600

```
myDatabase> db.customers.find({ "Payment.Total": 600 })
```

c. Find any record where price (Transaction.price) is between 300 to 500

```
myDatabase> db.customers.find({ "Transaction.price": { $gte: 300, $lte: 500 } })
```

d. Calculate the total transaction amount by adding up Payment.Total in all records

```
myDatabase> db.customers.aggregate([
...     {
...         $group: {
...             _id: null,
...             totalAmount: { $sum: "$Payment.Total" }
...         }
...     }
... ]);
```

MERN

4. Implement all CRUD operations on a File System using Node JS

1. Setup Node.js Project

First, initialize a new Node.js project if you haven't already:

mkdir file-system-crud

cd file-system-crud

npm init -y

2. Create index.js

3. Implement CRUD Operations

```
const fs = require('fs');
const path = require('path');

// Create a file
function createFile(filePath, content) {
    fs.writeFile(filePath, content, (err) => {
        if (err) {
            console.error(`Error creating file: ${err}`);
        } else {
            console.log(`File created successfully at ${filePath}`);
        }
    });
}

// Read a file
function readFile(filePath) {
    fs.readFile(filePath, 'utf8', (err, data) => {
        if (err) {
            console.error(`Error reading file: ${err}`);
        } else {
            console.log(`File content: ${data}`);
```

MERN

```
        }
    });
}

// Update a file
function updateFile(filePath, newContent) {
    fs.appendFile(filePath, newContent, (err) => {
        if (err) {
            console.error(`Error updating file: ${err}`);
        } else {
            console.log(`File updated successfully at ${filePath}`);
        }
    });
}

// Delete a file
function deleteFile(filePath) {
    fs.unlink(filePath, (err) => {
        if (err) {
            console.error(`Error deleting file: ${err}`);
        } else {
            console.log(`File deleted successfully at ${filePath}`);
        }
    });
}

// Example usage
const filePath = path.join(__dirname, 'example.txt');

// Create a file
createFile(filePath, 'Hello, World!');
```

MERN

```
// Read the file
setTimeout(() => readFile(filePath), 1000);  // Delay to ensure file is created

// Update the file
setTimeout(() => updateFile(filePath, '\nAppended content.'), 2000);  // Delay to ensure file is read

// Read the updated file
setTimeout(() => readFile(filePath), 3000);  // Delay to ensure file is updated

// Delete the file
setTimeout(() => deleteFile(filePath), 4000);  // Delay to ensure file is read again
```

4. Run the Script

```
node index.js
```

Explanation of the Code

1. **Create a file**: The createFile function uses fs.writeFile to create a new file or overwrite an existing file with the specified content.
2. **Read a file**: The readFile function uses fs.readFile to read the content of a file and log it to the console.
3. **Update a file**: The updateFile function uses fs.appendFile to append new content to an existing file.
4. **Delete a file**: The deleteFile function uses fs.unlink to delete a specified file.

Note

- **Error Handling**: Each function includes basic error handling that logs errors to the console. In a real-world application, you might want to handle these errors more gracefully.
- **Delays**: The setTimeout function is used to introduce delays between operations to ensure that each operation has time to complete before the next one starts. This is for demonstration purposes. In a production scenario, you would handle this more robustly, possibly with promises or async/await syntax.

5. a. Read the data of a student containing usn, name, sem, year_of_admission from node js and store it in the mongodb

b. For a partial name given in node js, search all the names from mongodb student documents created in Question(a)

Part a: Reading Student Data and Storing it in MongoDB

1. **Set up Node.js and MongoDB**:
   - Ensure you have Node.js installed.
   - Install MongoDB and run it on your local machine or use a cloud service like MongoDB Atlas.
   - Use the MongoDB driver or Mongoose to connect to MongoDB.
2. **Create a new Node.js project**:
   - Initialize the project.
   - Install necessary packages.
3. **Create scripts to read and store data**.

Step-by-Step Implementation:
Step 1: Initialize Node.js Project

```
mkdir student-mongo
cd student-mongo
npm init -y
npm install mongoose
```

Step 2: Create student.js for reading and storing student data

Create a file named student.js.

```
const mongoose = require('mongoose');

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/studentDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log('Connected to MongoDB');
```

MERN

```
}).catch((err) => {
    console.error('Error connecting to MongoDB:', err);
});

// Define a schema for the student
const studentSchema = new mongoose.Schema({
    usn: String,
    name: String,
    sem: Number,
    year_of_admission: Number
});

// Create a model from the schema
const Student = mongoose.model('Student', studentSchema);

// Function to add a new student
async function addStudent(usn, name, sem, year_of_admission) {
    const student = new Student({ usn, name, sem, year_of_admission });
    await student.save();
    console.log('Student added:', student);
}

// Example usage
addStudent('1MS17CS101', 'John Doe', 6, 2017).then(() => {
    mongoose.connection.close();
});
```

Step 3: Run the Script

node student.js

This script connects to MongoDB, defines a student schema, and adds a student record.

Part b: Search for Students by Partial Name

MERN

1. **Extend the script to include a search function**:
   - Use regular expressions to perform a partial name match.
   - Fetch the results from the MongoDB collection.

Step-by-Step Implementation:

Step 1: Update student.js to include the search functionality

```
const mongoose = require('mongoose');

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/studentDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log('Connected to MongoDB');
}).catch((err) => {
  console.error('Error connecting to MongoDB:', err);
});

// Define a schema for the student
const studentSchema = new mongoose.Schema({
  usn: String,
  name: String,
  sem: Number,
  year_of_admission: Number
});

// Create a model from the schema
const Student = mongoose.model('Student', studentSchema);

// Function to add a new student
async function addStudent(usn, name, sem, year_of_admission) {
  const student = new Student({ usn, name, sem, year_of_admission });
  await student.save();
```

MERN

```
    console.log('Student added:', student);
}
// Function to search students by partial name
async function searchStudentsByName(partialName) {
    const regex = new RegExp(partialName, 'i'); // 'i' for case-insensitive
    const students = await Student.find({ name: regex });
    return students;
}
// Example usage
(async () => {
    await addStudent('1MS17CS101', 'John Doe', 6, 2017);
    await addStudent('1MS17CS102', 'Jane Doe', 6, 2017);
    await addStudent('1MS17CS103', 'Johnny Smith', 6, 2017);

    const searchResults = await searchStudentsByName('John');
    console.log('Search results for "John":', searchResults);

    mongoose.connection.close();
})();
```

Step 2: Run the Script

```
node student.js
```

Explanation:

1. **Connection to MongoDB**: We connect to the local MongoDB instance using mongoose.connect.
2. **Schema and Model Definition**: We define a schema for the student collection and create a model from it.
3. **Adding Students**: The addStudent function adds a student document to the MongoDB collection.
4. **Searching by Partial Name**: The searchStudentsByName function uses a regular expression to search for documents where the name partially matches the input string.
5. **Example Usage**: We demonstrate adding students and searching for students by partial name. The search results are logged to the console.

MERN

6. Develop the application that sends fruit name and price data from client side to Node.js server using Ajax

1. Set up the Express.js server

First, ensure you have Node.js and npm installed. Then, create a new directory for your project and initialize it with npm:

```
mkdir fruit-app
cd fruit-app
npm init -y
```

Next, install Express.js:

```
npm install express
```

Create a new file called server.js and add the following code:

```
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json()); // Middleware to parse JSON bodies
app.use(express.static('public')); // Serve static files from the "public" directory

// Endpoint to receive fruit data
app.post('/add-fruit', (req, res) => {
  const { name, price } = req.body;
  console.log(`Received fruit data: Name - ${name}, Price - ${price}`);
  res.json({ message: 'Fruit data received successfully' });
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

MERN

2. Create the client-side HTML and JavaScript

Create a directory named public in the root of your project. Inside the public directory, create an index.html file and add the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Fruit App</title>
</head>
<body>
  <h1>Fruit App</h1>
  <form id="fruitForm">
    <label for="name">Fruit Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <label for="price">Price:</label>
    <input type="number" id="price" name="price" required><br><br>
    <button type="submit">Submit</button>
  </form>

  <script>
    document.getElementById('fruitForm').addEventListener('submit', function(event) {
      event.preventDefault();

      const name = document.getElementById('name').value;
      const price = document.getElementById('price').value;

      const data = {
        name: name,
        price: price
      };
```

MERN

```
    fetch('/add-fruit', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    })
    .then(response => response.json())
    .then(data => {
      console.log('Success:', data);
      alert('Fruit data sent successfully');
    })
    .catch((error) => {
      console.error('Error:', error);
    });
  });
 </script>
</body>
</html>
```

3. Run the application

Start your Express.js server by running the following command in your project directory:

```
node server.js
```

You should see a message in the console indicating that the server is running:

```
Server running at http://localhost:3000
```

4. Test the application

Open a web browser and navigate to http://localhost:3000. You should see a simple form to enter fruit name and price. Fill in the form and click the "Submit" button. The form data will be sent to the server using an Ajax request, and you should see a confirmation message in the browser and the server console log the received data.

MERN

7. Develop a React code to build a simple search filter functionality to display a filtered list based on the search query entered by the user.

1. Set up the React application

If you haven't already set up a React application, you can do so using Create React App:

```
npx create-react-app search-filter-app
cd search-filter-app
npm start
```

2. Create the SearchFilter component

Create a new file called SearchFilter.js inside the src folder and add the following code:

```
import React, { useState } from 'react';

const SearchFilter = () => {
  const [searchQuery, setSearchQuery] = useState('');
  const items = [
    'Apple',
    'Banana',
    'Orange',
    'Mango',
    'Grapes',
    'Pineapple',
    'Strawberry',
    'Blueberry',
    'Watermelon'
  ];

  const handleSearchChange = (event) => {
    setSearchQuery(event.target.value);
  };

  const filteredItems = items.filter(item =>
```

MERN

```
    item.toLowerCase().includes(searchQuery.toLowerCase())
  );

  return (
    <div>
      <h1>Search Filter</h1>
      <input
        type="text"
        placeholder="Search..."
        value={searchQuery}
        onChange={handleSearchChange}
      />
      <ul>
        {filteredItems.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
};

export default SearchFilter;
```

3. Use the SearchFilter component in your application

Open src/App.js and use the SearchFilter component:

```
import React from 'react';
import SearchFilter from './SearchFilter';

function App() {
  return (
    <div className="App">
      <header className="App-header">
```

MERN

```
      <SearchFilter />
    </header>
  </div>
 );
}

export default App;
```

Explanation

1. **State Management**: We use the `useState` hook to manage the state for the search query.
2. **Search Input**: An input field allows the user to enter a search query. The `handleSearchChange` function updates the state whenever the input changes.
3. **Filtering Logic**: The `filteredItems` array is derived by filtering the `items` array. The `filter` method checks if each item's name includes the search query (case-insensitive).
4. **Rendering the List**: The filtered items are displayed in an unordered list.

4. Running the application

After setting up the above code, your React application should be running. Open your web browser and navigate to `http://localhost:3000`. You should see a search input field and a list of items. As you type into the search field, the list should dynamically filter to show only the items that match the search query.

MERN

8. Develop two routes: find_prime_100 and find_cube_100 which prints prime numbers less than 100 and cubes less than 100 using Express JS routing mechanism.

```
const express = require('express');

const app = express();

function isPrime(num) {
  if (num <= 1) return false;
  for (let i = 2; i < Math.sqrt(num); i++) {
    if (num % i === 0) return false;
  }
  return true;
}

app.get('/find_prime_100', (req, res) => {
  const primes = [];
  for (let i = 2; i < 100; i++) {
    if (isPrime(i)) {
      primes.push(i);
    }
  }
  res.json({ message: 'Prime numbers less than 100:', primes });
});

app.get('/find_cube_100', (req, res) => {
  const cubes = [];
  for (let i = 1; i < 10; i++) {
    cubes.push(i * i * i);
  }
  res.json({ message: 'Cubes less than 100:', cubes });
```

MERN

```
});

app.listen(3000, () => console.log('Server listening on port 3000'));
```

Explanation:

1. We import express to create an Express application.
2. We define a function isPrime that checks if a number is prime.
3. We define two routes:
    - /find_prime_100: This route uses a GET method. It iterates through numbers from 2 to 99 (less than 100) and checks if they are prime using the isPrime function. If prime, it adds the number to a primes array. Finally, it sends a JSON response with a message and the list of prime numbers.
    - /find_cube_100: This route is also a GET method. It iterates through numbers from 1 to 9 (less than 10) and calculates the cube by multiplying the number by itself three times. It adds the cube to a cubes array. Finally, it sends a JSON response with a message and the list of cubes.
4. We listen on port 3000 for incoming requests.

This code defines two routes that handle GET requests and respond with JSON data containing the prime numbers and cubes less than 100. You can access these routes by sending GET requests to http://localhost:3000/find_prime_100 and http://localhost:3000/find_cube_100 in your browser or using tools like Postman.

Run

1. npm install express
2. node filename.js

MERN

9. Develop an authentication mechanism with email_id and password using HTML and Express JS (POST method)

Step 1: Set Up the Express Application

First, create a new directory for your project and initialize it with npm:

```
mkdir auth-app
cd auth-app
npm init -y
```

Install the necessary packages:

```
npm install express body-parser bcryptjs jsonwebtoken
```

Step 2: Create the Express Server

Create a file named server.js and add the following code:

```
const express = require('express');
const bodyParser = require('body-parser');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const app = express();
const port = 3000;

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

// In-memory user storage (for demonstration purposes)
const users = [];

// Route to handle user registration
app.post('/register', async (req, res) => {
  const { email_id, password } = req.body;

  // Check if user already exists
  const userExists = users.find(user => user.email_id === email_id);
  if (userExists) {
    return res.status(400).json({ message: 'User already exists' });
```

MERN

```
  }

  // Hash the password
  const hashedPassword = await bcrypt.hash(password, 10);

  // Save the user
  users.push({ email_id, password: hashedPassword });

  res.status(201).json({ message: 'User registered successfully' });
});

// Route to handle user login
app.post('/login', async (req, res) => {
  const { email_id, password } = req.body;

  // Find the user
  const user = users.find(user => user.email_id === email_id);
  if (!user) {
    return res.status(400).json({ message: 'Invalid email or password' });
  }

  // Check the password
  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid) {
    return res.status(400).json({ message: 'Invalid email or password' });
  }

  // Generate a JWT token
  const token = jwt.sign({ email_id: user.email_id }, 'your_jwt_secret', { expiresIn: '1h' });

  res.json({ message: 'Login successful', token });
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

MERN

Step 3: Create the HTML Form

Create a file named index.html in the same directory with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Authentication Form</title>
</head>
<body>
  <h2>Register</h2>
  <form id="registerForm">
    <label for="registerEmail">Email:</label>
    <input type="email" id="registerEmail" name="email_id" required>
    <label for="registerPassword">Password:</label>
    <input type="password" id="registerPassword" name="password" required>
    <button type="submit">Register</button>
  </form>

  <h2>Login</h2>
  <form id="loginForm">
    <label for="loginEmail">Email:</label>
    <input type="email" id="loginEmail" name="email_id" required>
    <label for="loginPassword">Password:</label>
    <input type="password" id="loginPassword" name="password" required>
    <button type="submit">Login</button>
  </form>

  <script>
    document.getElementById('registerForm').addEventListener('submit', async function(event) {
      event.preventDefault();
      const email_id = document.getElementById('registerEmail').value;
      const password = document.getElementById('registerPassword').value;

      const response = await fetch('/register', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ email_id, password })
      });

      const result = await response.json();
      alert(result.message);
    });

    document.getElementById('loginForm').addEventListener('submit', async function(event) {
      event.preventDefault();
      const email_id = document.getElementById('loginEmail').value;
      const password = document.getElementById('loginPassword').value;

      const response = await fetch('/login', {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ email_id, password })
    });

    const result = await response.json();
    if (result.token) {
      alert('Login successful');
    } else {
      alert(result.message);
    }
  });
 </script>
</body>
</html>
```

Step 4: Serve the HTML File

Modify your server.js to serve the index.html file:

```
const path = require('path');


// Serve the HTML file
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```

Step 5: Run the Application

Start your Express server by running the following command in your terminal:

```
node server.js
```

Step 6: Test the Application

Open your browser and navigate to http://localhost:3000. You should see the registration and login forms. You can now register a new user and log in with the credentials you provided.

MERN